

The image features a dark blue background with a fine, light blue grid pattern. Scattered throughout are numerous small, orange, elongated shapes resembling confetti or paper shreds. A thick orange border frames the entire image. In the top-left corner, there is a small orange icon consisting of a square with a horizontal bar and several vertical lines of varying heights. Centered in the image is the text "Github 활용" in a bold, orange, sans-serif font.

Github 활용



Git을 왜 사용할까?

- 이전의 기능을 다시 살리고 싶거나 전 버전으로 되돌리고 싶을 때
- 협업 시 내 코드와 다른 사람의 코드를 합치는 게 쉽고 충돌을 방지하기 쉬움





Git 이란?

- 코드 저장소
- Version Control System
- 분산 버전 관리 시스템



GitHub 란?

- 코드 저장소 (git)을 웹에 옮겨 놓은
원격 저장소

장점?

- 코드 복구 가능
- 협업가능

git 명령어



```
git status //git 상태창, 변경된 파일, 스테이징 파일
git restore --staged 파일명 //스테이징된 파일을 취소할 때
git log --all --oneline //commit 기록을 한 눈에 파악
git log --all --oneline --graph //commit 기록을 그래프로 그려줌
```

git add 저장소에 파일 추가



```
git add .           //디렉토리 안 모든 파일 추가  
git add 파일명      //특정 파일만 추가
```

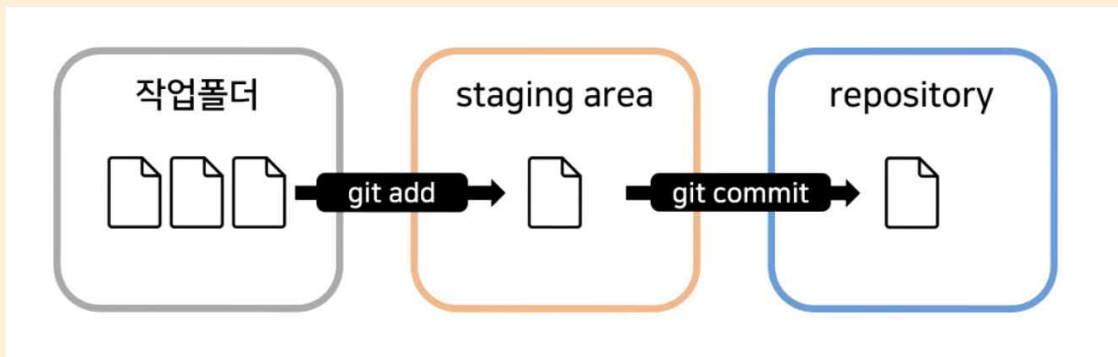
git commit

파일의 현재상태를 매일매일 기록

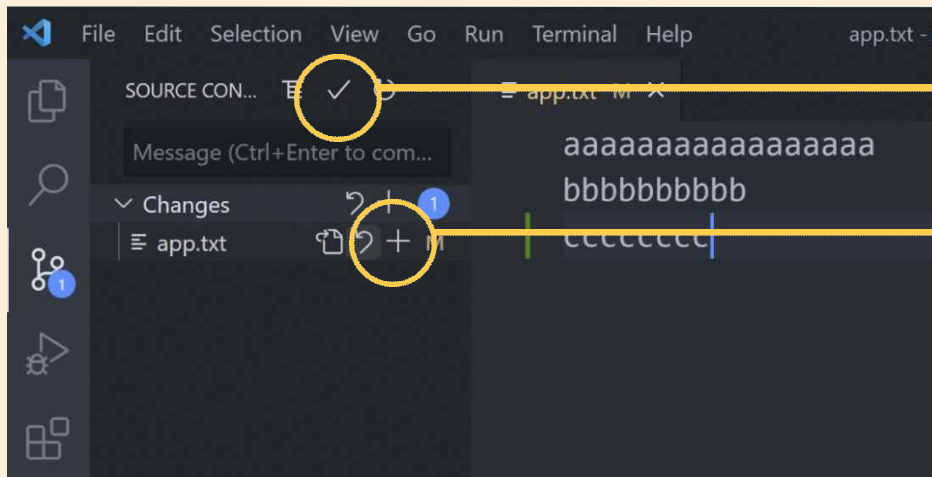


```
git commit -m "커밋메세지"
```

git add, commit의 과정



VScode 로 git을 사용하자!



git commit

git add

git diff

바로 전 commit과 현재 코드의 차이점을 비교



```
git diff                //전 커밋과 현재 코드 비교
git diff 커밋id         //과거의 특정 commit과 현재파일을 비교
git diff 커밋id1 커밋id2 //과거의 특정 commit 2개 간의 차이점 비교
```

git difftool

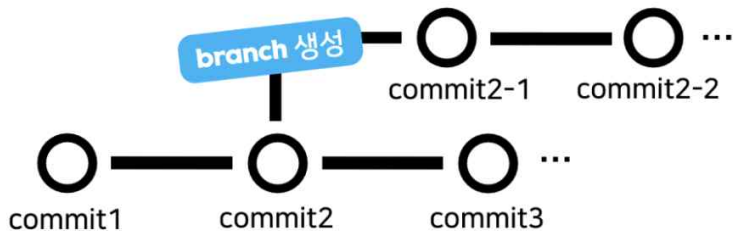
diff보다 비주얼이 좋음! 한 번 해보자



```
git difftool //전 커밋과 현재 코드 비교
git difftool 커밋id //과거의 특정 commit과 현재파일을 비교
git difftool 커밋id1 커밋id2 //과거의 특정 commit 2개 간의 차이점 비교
```


git branch

프로젝트의 복사본을 만들어서 개발할 수 있음



git branch

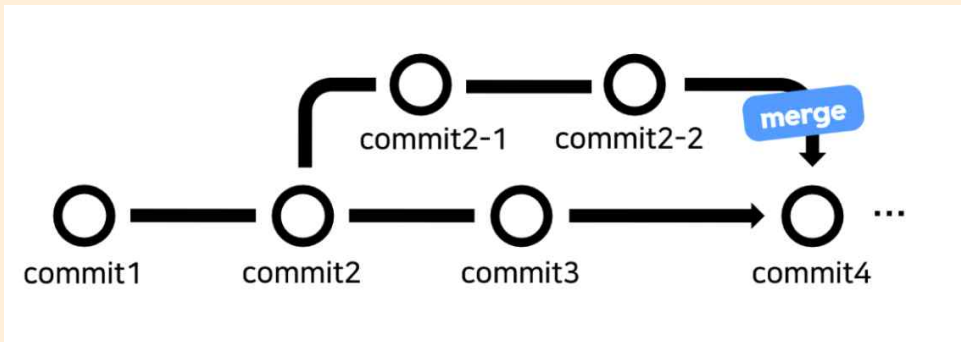
프로젝트의 복사본을 만들어서 개발할 수 있음



```
git branch 브랜치이름 //브랜치생성  
git switch 브랜치이름 //해당 브랜치로 이동
```

git merge

branch에서 개발한 내용을 main에 merge



git merge

branch에서 개발한 내용을 main에 merge



```
git switch main      //메인 브랜치로 이동  
git merge 브랜치명  //해당 브랜치를 병합
```

merge conflict

master 브랜치와 coupon 브랜치에서 같은 파일,
같은 줄을 수정했을 경우 발생

```
PS C:\Users\park\Desktop\작업중> git merge coupon
Auto-merging code.txt
CONFLICT (content): Merge conflict in code.txt
Automatic merge failed; fix conflicts and then commit the result.
```

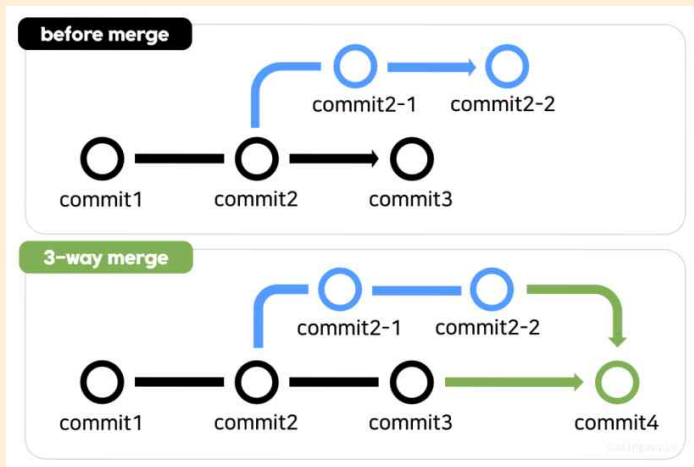

merge conflict 해결법



```
git add 파일명           //남길 파일을 staging  
git merge 브랜치명       //해당 브랜치를 병합
```

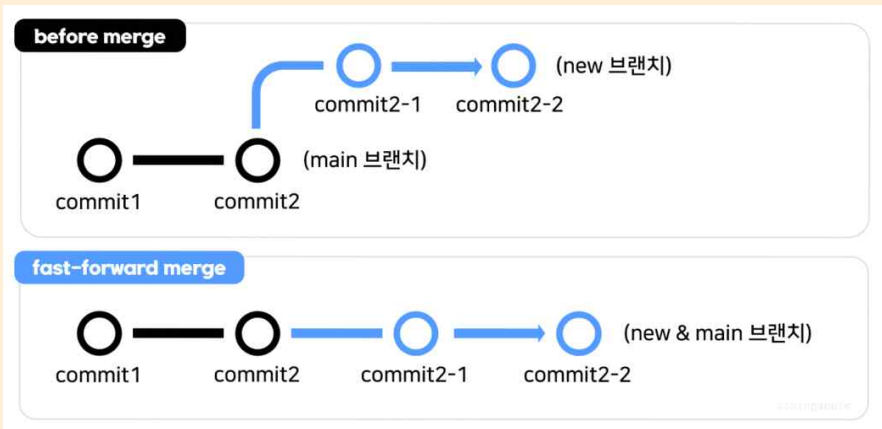
3-way merge

merge의 기본 방식



fast-forward merge

새로운 브랜치에만 commit 이 있고 기준이 되는 브랜치에는 신규 commit 이 없는 경우



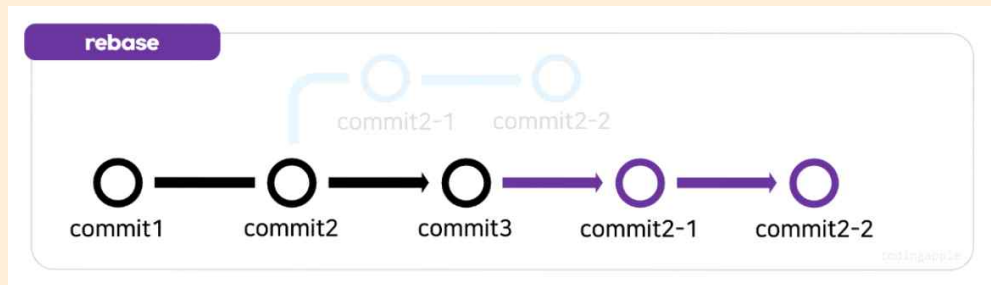
branch 삭제




```
git branch -d 브랜치이름 //병합이 완료된 브랜치 삭제  
git branch -D 브랜치이름 //병합하지 않은 브랜치 삭제
```

rebase

브랜치의 시작점을 다른 commit으로 옮기기



rebase & merge



```
git switch 새로운브랜치    //새로운 브랜치로 이동
git rebase main            //브랜치가 main브랜치 끝으로 이동

git switch main            //main 브랜치로 이동
git merge 새로운브랜치    //merge
```

git restore 파일 되돌리기



```
git restore 파일명           //최근 commit된 상태로 되돌리기  
git restore --source 커밋id 파일명 //파일이 특정 커밋 아이디 시점으로 복  
git restore --staged 파일명  //특정 파일을 staging 취소
```

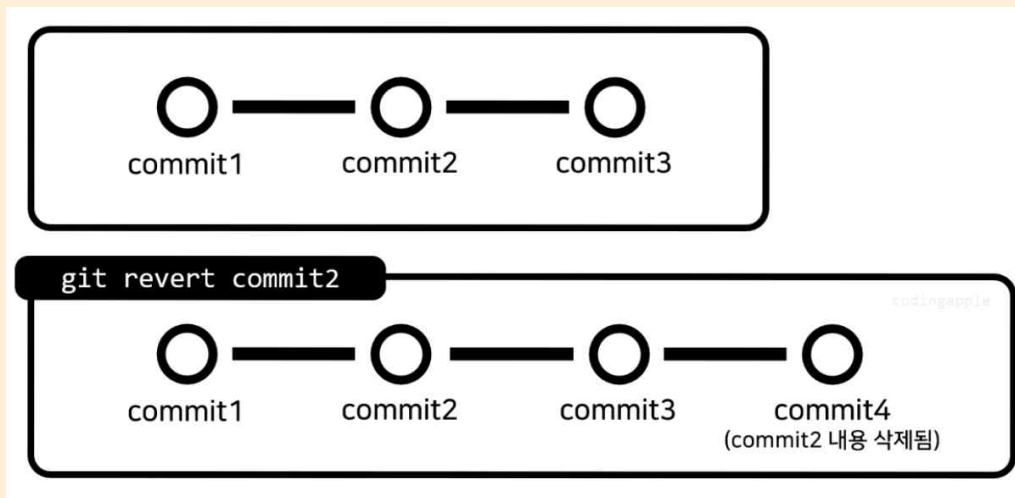
git revert

파일 되돌리기



```
git revert 커밋아이디 //커밋아이디에서 일어난 일만 취소
```


git revert 삭제한 내용의 commit이 새로 생김!



git revert

- revert 할 때 동시에 여러개의 commit id 입력가능
- 최근 했던 commit 1개만 revert하고 싶으면 git revert HEAD 하면 편리
- merge 명령으로 인해 새로 만들어진 commit도 revert 가능

그냥 시간을 전부 되돌리고 싶다면 ...



협업할 때는 사용 금지..

- untracked 파일들은 (git add 안해놓은 파일들은) 사라지지않고 유지
- git clean 명령어 찾아서 쓰면 untracked 파일들도 다 지울 수 있다!

첫 번째 : 내 코드를 올려보자

1) repository란?

2) GitHub에서 repository만들기

3) 거기에 내 코드 올리기

1) repository란?

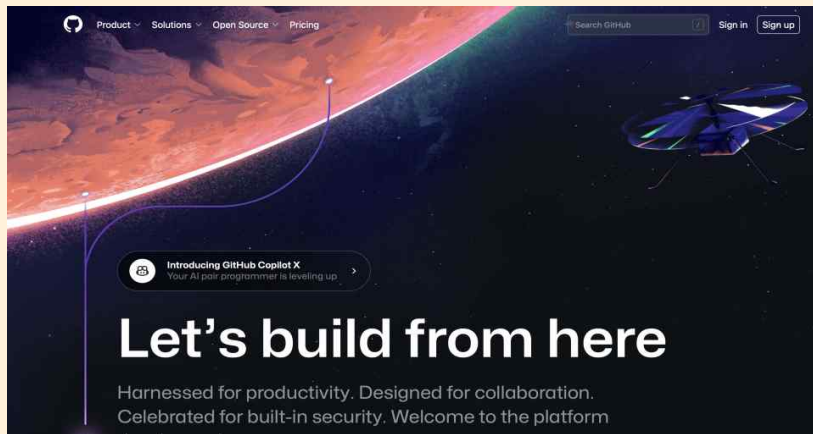
- 파일 버전 저장소
- 온라인repository의 장점?
 - 1) 내 컴퓨터 고장나도 파일은 안전
 - 2) 협업가능



1) GitHub에서 repository만들기

step 1

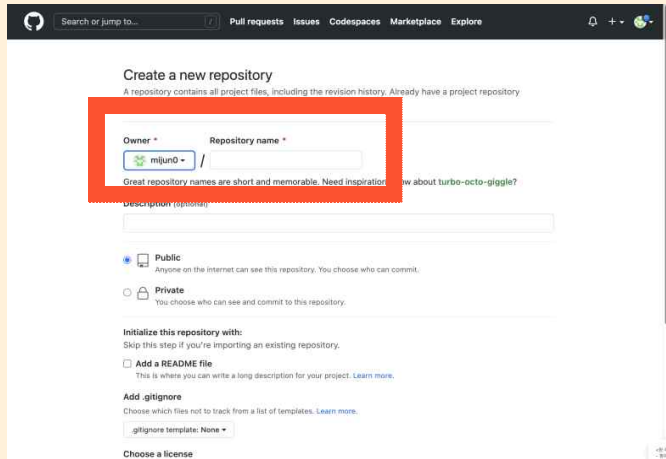
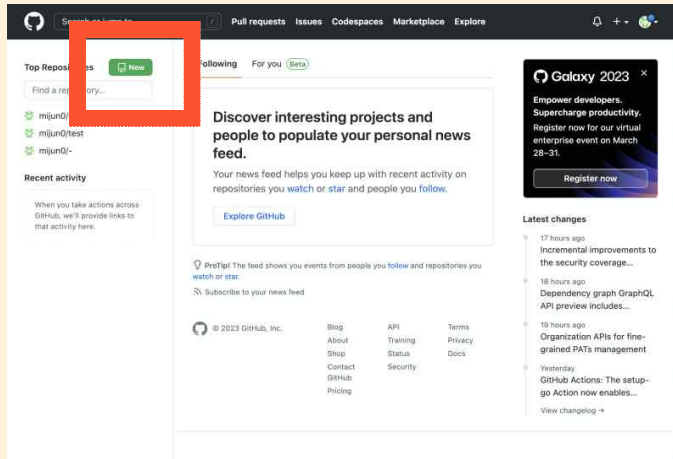
깃허브(<https://github.com/>) 가입 후 로그인



1) GitHub에서 repository만들기

step 2

깃헙에서 new repository 만들기



2) GitHub에 내 코드 올리기

step 3

로컬파일을 GitHub에 백업하기

1. 작업파일 만들고 터미널에서 `git init` (=repository생성)

- GitHub에서는 메인브랜치 이름 변경 권장. `git branch -M main` 입력

2. 테스트로 아무 파일 만들고 `add, commit`을 해봅시다

- `touch 파일명.확장자` (= 파일만들기)
- `vim 파일명.확장자` (= 파일에 뭐 쓰기)

vim 설명 : i 누르면 인서트모드, esc누르면 노멀모드, :wq하면 저장하고 나감)

2) GitHub에 내 코드 올리기

step 3

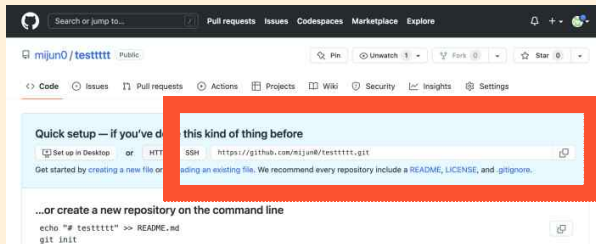
로컬파일을 GitHub에 백업하기

3. push : 로컬저장소에서 원격저장소로 옮깁니다

- 작업폴더에서 터미널키고 `git push -u` 원격저장소주소 브랜치명
- -u 는 기억하라는 의미이기 때문에 이 다음부터 `git push`만 해도 됩니다
- 원격저장소 주소는 GitHub에 있어요
- 원격저장소 줄여쓰기

: `git remote add` 변수명 레포주소

: 보통 변수명으로 origin 사용합니다



두 번째 : github로 협업하기

- 1) git clone으로 다운로드
- 2) git pull으로 업데이트

소스코드 다운로드?

Q. 내가 올린 소스코드를 협업하는 사람이 쓰려면?

A1. GitHub에서 코드 다운로드 (Code>Download ZIP)

A2. `git clone` 레포지토리주소 로 다운로드

1) git clone

step 0

실습이니까 '팀원' 작업폴더를 새로 만듭시다.

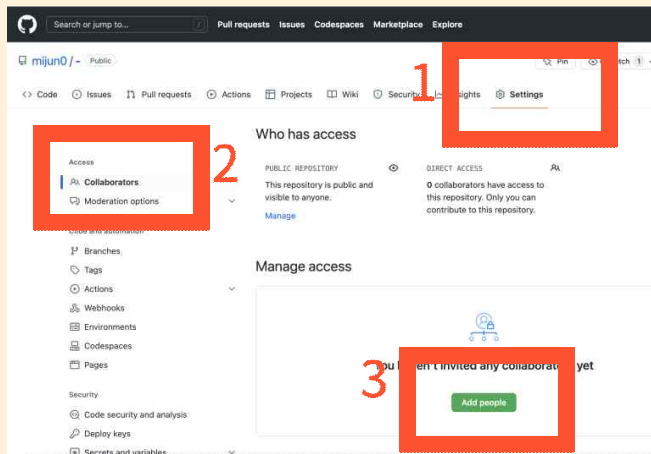
step 1

팀원의 터미널에서 `git clone` 레포지토리주소 입력. 끝.

1) git clone

step 2

협업이니까 팀원도 push할 수 있게 해주세요.



- Settings > Collaborators
- Add people 에서 팀원 ID 입력

git push 외양도...

```
(base) mijunjung ~/Desktop/ㄱ | ㅅ ㄱ main
> git push
To https://github.com/mijun0/-.git
 ! [rejected]        main -> main (non-fast-forward)
error: failed to push some refs to 'https://github.com/mijun0/-.git'
hint: Updates were rejected because the tip of your current branch is behind
hint: its remote counterpart. Integrate the remote changes (e.g.
hint: 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
(base) x mijunjung ~/Desktop/ㄱ | ㅅ ㄱ main
```

- 로컬(내컴퓨터)과 원격(깃허브)가 다르다면 git push가 되지 않아요
- 원격저장소가 업데이트가 되었기 때문에 로컬도 최신으로 만들어주어야해요

2) git pull

현재 원격저장소 내용을 로컬로 가져오기

역할 : 로컬에 없는 신규 commit을 가져와 병합합니다

1) 로컬을 업데이트하기

- 터미널에 `git pull` 레포지토리주소 입력
- 특정 브랜치만 가져오기 : `git pull 레포지토리주소 브랜치명`

2) git push 하려던 것 하기

세 번째 : branch로 협업하기

branch 만드는 이유?

1. 많은 사람들과 협업 > 수많고 끊임없는 push?
2. main에 코드를 짜다가 망치면?

**branch 만들어서 작업하고
merge 하는게 안정적!**

branch 로 협업하기

step 1

브랜치 만들기

1. `git branch` 브랜치명 (= 브랜치 만들기)
2. `git switch` 브랜치명 (= 해당 브랜치로 이동)
 - 여기서 파일 만들고 코드를 짭니다

step 2

만든 브랜치 원격저장소에 올리기

`git push` 레포지토리주소 브랜치명

branch 로 협업하기

step 3

브랜치 병합하기

참고1 : 다른 브랜치들과 내용이 겹쳐 병합이 되지 않는 경우, GitHub에서 어떤 부분들을 살리고 죽일지 골라 병합할 수 있습니다.

참고2 : merge 종류는 3가지가 있습니다

- create a merge commit = 각 브랜치들의 커밋내용이 모두 남습니다
- squash and merge = 커밋내용이 하나로 합쳐져 신규 커밋을 생성합니다
- rebase and merge = 1과 2 믹스? 커밋내용은 남고 결과는 스쿼시와 비슷

The image features a dark blue background with a fine, light blue grid pattern. A thick orange border frames the entire scene. Scattered throughout the background are numerous small, orange, elongated shapes resembling confetti or paper shreds. In the top-left corner, there is a small orange icon consisting of a square frame with a series of vertical bars of varying heights inside. Centered on the grid is the Korean text '감사합니다' in a bold, orange, sans-serif font.

감사합니다